



THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office


November 09, 2004

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: 60/507,185
FILING DATE: *September 29, 2003*
RELATED PCT APPLICATION NUMBER: PCT/US04/32296

Certified by




Jon W Dudas

Acting Under Secretary of Commerce
for Intellectual Property
and Acting Director of the U.S.
Patent and Trademark Office

BEST AVAILABLE COPY

09/29/03



Express Mail mailing label no. EL 584772861 US

Date of Deposit: September 29, 2003

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to Mail Stop Box Provisional Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

By:

Rhonda Dunn

Attorney Docket No. HAND0003PR

17302 U.S. 1185

60/507185

092903

IN THE U.S. PATENT AND TRADEMARK OFFICE

Provisional Application Cover Sheet

Mail Stop Box Provisional Patent Application
 Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450

Sir:

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(b)(2).

INVENTOR(S)/APPLICANT(S)

Last Name	First Name	Residence (City and Either State or Foreign Country)
Hamilton	Eric	Los Gatos, California, USA
Page	Carl	San Francisco, California, USA
Dolgoborodov	Alexey	St. Petersburg, Russia
Tikhonov	Anton	St. Petersburg, Russia
Semenyuk	Vladimir	St. Petersburg, Russia

Title of the Invention

HAND HELD AUDIO VIDEO DISPLAY DEVICE

Correspondence Address

Glenn Patent Group
 3475 Edison Way, Suite L
 Menlo Park, CA 94025

Telephone No. (650) 474-8400

Enclosed Application Parts (check all that apply)

(X) Specification and Drawing(s)
 Number of Pages 28

() Small Entity Statement - Small Business

Filing Fee and Method of Payment

☒ \$80.00 for Small Entity

☐ \$160 for Large Entity

The Commissioner is authorized to charge the filing fee of \$80.00 and any additional fees or credit any overpayment to Deposit Account No. 07-1445 (Order No. HAND0003PR). A copy is enclosed for this purpose.

Respectfully Submitted,

Michael A. Glenn
 Reg. No. 30,176

Customer No. 22862

HHE Software Inventions

1. Methods for complexity reduction of video decoding
2. Specific methods for implementing an MP3 decoder using fixed point arithmetic
3. Specific method for implementing fast YcbCr to RGB conversion
4. Method for encapsulating a video stream and MP3 audio stream in an AVI file
5. Method for storing menu navigation and DVD subpicture information on a memory card
6. Specific method for synchronizing the audio and video streams
7. Method for encrypting keys used for decryption of multimedia data.
8. Certain aspects of the HHE User Interface (UI)

From: "Carl Page" <carlp@findpage.com>
Date: Mon Sep 15, 2003 09:47:07 US/Pacific
To: <michael@glenn-law.com>
Cc: <erich@hheld.com>, "garrettl. cecchini"
<garrett@hheld.com>
Subject: Re: HHE hardware documents

mb tech

TODO: LIST ANTIPIRACY TECHNOLOGIES TO BE USED IN
MOCKINGBIRD

May include

- Encrypted content
- Decrypt Key locked in hardware.
- Use of SDCard? encryption key. (This key may be servicable but has been proven breakable.) A slashdot forum is [here](#)
- DVD technologies. Description The new CSS2 system was designed for DVD-Audio, but will now be used by all DVD formats. Hase says it is much more robust, offering an almost limitless number of keys, compared with the six offered by CCS mark one: As soon as the software is broken, we can issue a new key, so the system is more efficient and practically undefeatable. The new copy protection system was completed by the end of December 1999, and has two main components: CPRM (Copy Protection for Recordable Media) and CPPM (Copy Protection for Pre-recorded Media). The former will be used for formats such as DVD-RAM and DVD-RW. The complete system is now called 4C, after the four companies which developed it.. Music companies are now testing 4C with their golden-eared specialists, to see whether it has an adverse effect on sound quality.
- Macronix has a casual level of security Palm uses for their MMC cards that is royalty free. See [MMCVendors](#)

A serious reason for preferring Mockingbird players to general purpose PDA's will be their resistance to software and content piracy.

An organization lobbying to criminalize content sharing and piracy is of course, the RIAA <http://www.google.com/search?q=riaa> Their website seems to be the target of hackers yet again tonight, so I can't get in. It has been defaced several times, most recently with nicely written position statements more reasonable what they normally espouse. <http://www.riaa.org>

Microsoft also has an "astroturf roots organization" supporting its antipiracy effort, the Software Publisher's Association. <http://www.spa.org/piracy/>

One organization in opposition to the some of the RIAA's efforts strong Bay Area Support, in part from prominent recording artists, is the <http://eff.org>

Our approach will be to open up new avenues of legal distribution on an attractive player.

We will not bank on legislated antipiracy efforts being successful at eliminating piracy. m

From DigitalMediaWire? 4/2/2003 (Mountain View, Calif.) The Internet Streaming Media Alliance (ISMA), a consortium of companies working to develop a universal, open standard for Internet streaming media, announced on Wednesday it is releasing for peer review a new content protection specification. Mountain View, Calif.-based ISMA, whose members include Apple, Philips, AOL Time Warner and Sony, said finalization of the new content protection specification is expected in June. The specification provides a single, end-to-end encryption scheme for streaming media and file downloading that can easily integrate with different key and rights management solutions and licensed content protection devices. <http://www.isma.tv>

MPEG4 complexity reduction solutions

In order to reduce the complexity of MPEG4 decoding the following four solutions have been introduced:

1. Disabling of intra prediction of AC coefficients

Intra prediction of AC coefficients is not made. Flag that indicates the need for AC prediction has been eliminated from the bitstream.

2. Disabling of motion compensation rounding control

Rounding control is disabled. Constant additions are used during averaging: 0 for averaging of two values and 1 for averaging of four values. Rounding bit has been eliminated from the bitstream.

3. Combination of VLC decoding and dequantization in one step

Dequantization of the coefficient is made right after decoding of its variable length code. Speed-up is possible due to exclusion of zero coefficients from dequantization process.

4. Simplification of inverse discrete cosine transformation with the use of significance map

Significance map is used to store the positions of last nonzero coefficients in each row/column of discrete cosine transformation block. Significance map is filled during VLC decoding. Knowing the number of last nonzero coefficient in row/column it is possible to simplify the inverse discrete cosine transformation for this particular row/column. Two different versions of inverse discrete cosine transformation are provided: one - for rows/columns of 8 coefficients and one for rows/columns of 3 coefficients. Note, that when all coefficients in row/column are zero coefficients, inverse transformation should not be made at all.

Description of fast "YUV to RGB555" conversion

In order to speed-up the color conversion routine conversion table is used. Table index is calculated as a function of 3 colors in YUV format:

$$\text{Index} = ((U \gg (8 - \text{BITS_U})) \ll (\text{BITS_Y} + \text{BITS_V})) + \\ ((V \gg (8 - \text{BITS_V})) \ll (\text{BITS_Y})) + \\ (Y \gg (8 - \text{BITS_Y})),$$

where Y, U and V - 8-bit color components in YUV format; BITS_Y, BITS_U, BITS_V - numbers of significant bits for each color: Y, U and V.

Number of indexes is $(1 \ll (\text{BITS_Y} + \text{BITS_U} + \text{BITS_V}))$. Conversion table cell represents color in RGB555 format that corresponds to color in YUV format. Size of the cell is 2 bytes (high-order bit is unused). Therefore, size of the table is number of indexes * 2, that is: $(1 \ll (\text{BITS_Y} + \text{BITS_U} + \text{BITS_V} + 1))$.

Number of significant bits for Y color component must be greater than number of significant bits for U and V components, because Y color component contains more useful information for human visual perception. Currently the following significant numbers are used:

$$\begin{aligned} \text{BITS_Y} &= 7 \\ \text{BITS_U} &= 5 \\ \text{BITS_V} &= 5 \end{aligned}$$

Color conversion table is organized in the manner that can help to avoid cash misses during conversion of image in YUV 4:2:0 format. In YUV 4:2:0 format for each chrominance pixel there are 4 luminance pixels. A fact that index depends on Y component less than on U and V components makes data cash misses infrequent.

ZVue file formats

The file format for storing ZVue media comes from the way the navigation system, the graphics system and the decoding engines are designed.

We assume that media containing video/audio streams is organized in chapters, associated with navigation scripts and can optionally carry a custom decoding engine.

The media should be FAT16-formatted, and the content organized in files. All data is stored in the root folder, other folders are ignored if present.

Files on the media are:

- **"config"** main configuration file for the media that specifies the media type (currently only two types are supported: ZVUE-VIDEO and FIRMWARE), the main navigation script file name, the decoding engine to use (a custom one can go on the media, the default one resides in a flash).
- **"*.nav"** navigation scripts for video chapters
- **"*.avi"** video/audio streams
- **"*.mnu"** menu files, that describe menu representation and functionality by specifying subpictures for menu items, pointers to chapters etc.
- **"*.bmp"** menu subpictures that are MS Windows 16-color compressed bitmaps. Colors {0,0,0} and {255,255,255} are reserved for transparent.

File types that are not supported but can be added later

- **"*.mp3"** audio only streams.
- **"*.jpg", "*.jpeg"** jpeg images (for browsing digital photos from SD card, or to use as menu background etc.).

Configuration file

This is a plain text ASCII file in either windows (CR/LF) or unix (CR) format.

- A semicolon ';' starts line comment
- Commands are : <key> = <value>. Spaces are allowed. If value contains spaces, it should be enclosed in double quotes ("")
- Empty lines are ignored

Some keys may not be defined. The default semantics is applied then

Key	Value	Defaults
application	Filename of the executable to use as a decoder	Use internal decoder from the flash
start	Filename of main menu navigation script (the navigation script that is run first)	Runs first *.nav file found on the media
type	Media content type	ZVUE-VIDEO
encryption_key	Encrypted checksum to verify the firmware	-

version	Firmware version	0
---------	------------------	---

Type=ZVUE_VIDEO

Notifies the boot loader that this card stores video content. If Application tag is present, the boot loader loads it to memory and runs there. If not, the boot loader loads application from the flash.

Type=MP3 (not supported)

Notifies the boot loader that this card stores mp3 tracks. If Application tag is present, the boot loader loads it to memory and runs there. If not, the boot loader loads application from the flash. The application runs as a standard MP3 player. *(not supported)*

Type=PHOTO (not supported)

Notifies the boot loader that this card stores JPEG images. If Application tag is present, the boot loader loads it to memory and runs there. If not, the boot loader loads application from the flash. The application runs in slide-show mode.

Type=FIRMWARE

Notifies the boot loader that this card stores new media driver. The loader checks zveu.axf file from the card with encrypted checksum *encryption_key* and then burns it to the flash. It also checks the version against current and notifies user if it is older.

Limitations of the 1st version:

- file format restricted to windows (CR/LF)
- Last line should end with CR/LF
- Quoted values are not supported

AVI file

We use standard windows AVI format for streaming the videos. The file should contain one video stream, coded with HHE video encoder (FOURCC=HHE0) and/or one audio stream, coded with any MP3 driver (wFormatTag=0x0055). When using B-frames, they should be put into separate AVI chunks. Usually, it requires some post processing as the VFW drivers usually are not capable of producing it. We can create a more sophisticated solution later.

The video bitstream format spec will go in a separate document

The audio bitstream format complies with ISO CD 11172-3 document.

Current limitations

- both audio and video streams should always be present
- MP3 limitations are: MPEG1, Layer3, sample rate=44100, emphasis ignored.

Navigation script file

Navigation scripts specify the semantics of player buttons for the specific chapter, the avi stream and subpictures to use and the actions to perform. The navigation script is a text file, with navigation commands represented on separate lines. Commands are case-sensitive.

Commands are : <key> = <value>. Spaces are allowed. If value contains spaces, it should be enclosed in double quotes ("")

Command set:

stream = <avi-file>

Specifies an AVI file associated with this script

next = <scriptname>

Specifies a chapter that runs after this one is ended.

previous = <scriptname>

Specifies a chapter to start on REW.

A semicolon at first position starts line comment.

If it is the first chapter in a chain, **previous** should not be present.

If it is the last chapter in a chain, **next** should not be present.

Menu file

Menu file is a text file that specifies the menu appearance and functionality. Commands should start at the beginning of each line, command arguments follow on the same line, any number of white space characters (' ', '\t') can be used as a separator. Menu contains a background image (stored in AVI), a number of static bitmaps over the background and a number of menu items associated with video chapters. Command arguments are either filenames or numbers, filenames should be put in double quotes. All arguments are obligatory.

A semicolon at first position starts line comment.

Command set:

background *avi-file*

specifies an AVI (usually of one frame) that contains menu background, The AVI file is played on the screen, and the last frame of that AVI is used as a background for menu.

static *bitmap x y transparency*

specifies a static bitmap displayed over the background image. *x, y* specify the bitmap offset from the top left corner; *transparency* is a number from 0 to 255 that specifies the transparency (0 means transparent, 255 means solid)

item *bitmap_0 x y transparency bitmap_1 x y transparency navig_script menu*

describes menu item. *bitmap_0* is displayed for a selected item, *bitmap_1* is displayed for deselected ones, *x, y* and *transparency* following a bitmap name specify its position and transparency. *navig_script* specifies the script to start when this menu item is executed, if "", this means a submenu should be run, specified in *menu* argument. *menu* sets new menu for the script to run, or a submenu to run, if script name is not specified. If it is "", current menu is used.

Software and hardware specs

Flash specifications

Requested: SPB team

Author: J. Lewis

Implementation: J. Lewis

Status: *not implemented*

On startup, the program is copied from the flash, addresses 0x80000—0xFFFFF to SDRAM, addresses from 0x08080000 — 0x080FFFFF. The program control goes to address 0x08080000.

To write flash, the API entries are used according to flash.h file:

```
HHE_RESULT InitializeFlash(void);
HHE_RESULT EraseSector(unsigned char sectorToErase);
HHE_RESULT FlashWrite(unsigned char* dataToWrite,
                      unsigned long startingLocation,
                      unsigned long numberOfBytesToWrite) ;
HHE_RESULT ShutdownFlash(void) ;
```

MMC driver specifications

Requested: E. Hamilton

Author: A. Dolgoborodov

Implementation: J. Lewis

Status: *partially implemented*

1. We call some function to notify the driver we need some file. The driver should start buffering this file in background and return with a minimal delay.
2. File read operations copy buffered data. We guaranty all access to files is in address increasing order, but skipping is possible on rewind. Skipping on rewind is by some relatively small portions (several kilobytes, or maybe 10-20 kbytes).
3. As an exclusion from the rule above, we need to read index from the end of an AVI file with some given offset, and then return to the beginning of the file, but here we can allow some delay, it is not critical.
4. The average throughput for read operations should be 1MBit or more
5. The processor load for managing MMC loads (time spend inside read() function) shouldn't be more than 5% at 500 kbps (the less the better!)
6. On rewind, when we skip data in the stream, small stalls are allowed. Additional processor load is also allowed.

Boot loader spec

Requested: SBP team

Author: A. Dolgoborodov

Implementation: A. Budkin

Status: *not implemented*

Boot loader address space:	0x08000000 — 0x08080000
System memory address space:	0x08000000 — 0x08080000
Program location in flash:	0x10080000 — 0x100FFFFF
Interrupt vector location:	0x00000000
sys_info location:	??

The boot loader resides in lower addresses in flash, along with a set of drivers. On startup, the boot loader unloads drivers to system, then runs start application.

The start application checks for SD/MMC card in the slot. If there is no card, it asks user to insert a card. If card type is FIRMWARE it goes to **firmware update mode**. Otherwise, it goes to **run mode**. The type of card is determined from file "config" (see file format specification).

In **firmware update mode**, the application should decrypt a program, check the checksum, and if it is Ok, write it to flash. It also checks the build number and notifies user if it is older than the current version. The flash write operation should be verified. After finished, start application notifies user to take the SD-card off and restart. The encryption key for applications is stored in a boot loader code.

In **run mode**, the start application checks if the media contains an application. If yes, decrypts it, checks the checksum and launches it. Otherwise starts an application from the flash.

Boot loader goes with hardware drivers that are necessary for it function: screen, keyboard, decryption, SD/MMC and file system drivers, (font generator or graphics engine). It exports vector of API entries and also system info structure, so that application could use it:

```
struct sys_info {
    t_ui32      ver_hw    // hardware version
    t_ui32      ver_API   // API version
    t_ui32      user_mem_start // User address space
    t_ui32      user_mem_end   //
    t_ui32      nd_user_mem_start // Legal address space if
    t_ui32      nd_user_mem_end   // application doesn't use
                                // hard-coded API

    struct api_entries vf_table;
}
struct api_entries {
    void * func0;
    void * func1;
    void * func2;
    ...
    void * reserved0;
    void * reserved1;
};
```

MP3 Audio decoding driver spec

Requested: SBP team

Author: A. Dolgoborodov

Implementation: A. Dolgoborodov, A. Budkin

Status: *done*

Audio driver decodes a stream frame-by-frame, assuming frames are passed to it sequentially. The output should always be stereo@44100. Whenever frames are passed to the driver in discontinuous order, the application should reset it explicitly.

API calls used for MP3 decoder:

`result_t mp3_init(void)`

Initializes the driver. This function is called once, on decoder start.

`Result_t mp3_shutdown(void)`

Shuts down the driver. This function is called once, on decoder shutdown.

`result_t mp3_reset(void)`

Notifies the driver to discard buffers filled from previous frames. This function should be called prior to decoding the first frame and every time the continuity of data passed to the decoder is broken.

`result_t mp3_decode_frame(t_ui8* frame, sample_t * out)`

Decodes one frame. Returns number of bytes written to the output buffer (should always be 9216).

HHE Video decoding driver spec

TBD

UI spec

See file UI.doc

Encryption spec

Requested: SBP team

Author: E. Hamilton

Implementation: SPB team

Status: *not implemented*

The blowfish 64-bit symmetric encryption is used for HHE software and multimedia data.

The HHE software decryption is stored inside the boot loader code. The executable that goes on the MMC should be decrypted and check sum verified .

For the multimedia data, the encryption is applied separately to video and audio streams inside an AVI file. The encryption of multimedia data is optional. The encrypted streams have different FOURCC and wFormatTag. The encryption key is stored in the application.

Each video/audio chunk is encrypted from from the 1st byte, in portions of 8 bytes (64 bits). If the chunk size is not a multiple of 8, the extra bytes are not encrypted.

USB driver spec

Requested: SBP team

Author: E. Hamilton

Implementation: SPB team

Status: *not implemented*

The USB driver exports FAT file system on the user SD/MMC card to PC. The file system is exported RW. The main aim of USB interface is to allow user download MP3 and JPEG stills to the device. The file system on HHE ROM MMC is not exported for security reasons.

Authoring tools spec

Requested: SBP team

Author: E. Hamilton

Implementation: SPB team

Status: *not implemented*

The authoring tools should provide a minimal functionality:

- Encodes video and audio into final AVI format
- Able to encode multiple video/audio chapters
- Supports one top menu with multiple menu items oriented vertically
- Supports still picture menu background
- Each menu item associated with a different chapter
- Supports encryption of video+audio using specified key

Engineering solutions used in HHE fast fixed-point implementation of MPEG-1 Layer 3 decoding algorithm.

1. General remarks on operations with fractional values for fixed point arithmetic

To represent data in fixed point operations, we use the following transformation:

$$u = \text{Fix}(u_{\text{float}}) = (\text{int})(u_{\text{float}} * (2^{n\text{BitsFraction}}) + 0.5), \quad (1.1)$$

where $n\text{BitsFraction}$ - number of bits for fractional part, value 0.5 is used for rounding.

The following values of $n\text{BitsFraction}$ are used:

- 24 for signal samples (representation 32.24),
- 24 or 15 for constant coefficients (representation 32.24 or 32.15)

Let

$$y_{\text{float}} = x_{\text{float}} * c_{\text{float}},$$

where $x_{\text{float}}, c_{\text{float}}$ - some variables (c_{float} is usually a constant).

Then, in the case of 32.24 data representation,

$$\begin{aligned} x &= (\text{int})(x_{\text{float}} * (2^{24}) + 0.5), \\ c &= (\text{int})(c_{\text{float}} * (2^{24}) + 0.5), \\ y &= (x * c) \gg 24. \end{aligned}$$

As we use 32-bit integer operations, it is necessary to avoid overflow in calculation of product $x * c$.

For this purpose, we represent data as a sum of high and low parts:

$$\begin{aligned} u &= u_{\text{Low}} + (u_{\text{High}} \ll 12), \text{ where} \\ u_{\text{High}} &= u \gg 12, \\ u_{\text{Low}} &= u - (u_{\text{High}} \ll 12) = u \& 0x00000FFF \end{aligned}$$

Thus, we have

$$y = (x * c) \gg 24 = ((x_{\text{Low}} + (x_{\text{High}} \ll 12)) * (c_{\text{Low}} + (c_{\text{High}} \ll 12))) \gg 24$$

This expression can be rewritten as

$$y = x_{\text{High}} * c_{\text{High}} + ((x_{\text{Low}} * c_{\text{High}} + c_{\text{Low}} * x_{\text{High}}) \gg 12) + ((x_{\text{Low}} * c_{\text{Low}}) \gg 24)$$

To speed up the multiplication, we can remove small parts from this sum. In our implementation, we distinguish 3 different levels of precision, any of them can be chosen at compile time. The simplifications used for multiply operation in each mode are as follows:

For high precision

$$y = x_{\text{High}} * c_{\text{High}} + ((x_{\text{Low}} * c_{\text{High}} + c_{\text{Low}} * x_{\text{High}}) \gg 12) \quad (1.2)$$

For medium and low precision:

$$y = x_{\text{High}} * c_{\text{High}} + ((x_{\text{Low}} * c_{\text{High}}) \gg 12) \quad (1.3)$$

For 32.12 representation of constant coefficients,

$$c = (\text{int})(c_{\text{float}} * (1 \ll 12) + 0.5).$$

The simplified multiplication on constant coefficients in 32.24 representation can be implemented as

$$y = ((x \gg 6) * c) \gg 6, \quad (1.4)$$

in assumption that

$$|c_{\text{float}}| < 1$$

If

$$1.0 < |c_{\text{float}}| < 2.0,$$

the multiplication is performed as

$$y = ((x \gg 6) * c) \gg 5 \quad (1.5)$$

where

$$c = (\text{int})(c_{\text{float}} * (1 \ll 12) + 0.5),$$

In a similar way, if

$$1.0 < |c_{\text{float}}| < (1 \ll q),$$

it's possible to use approximate multiplication in a form

$$y = ((x \gg 6) * c) \gg (6 - q) \quad (1.6)$$

Then

$$c = (\text{int})(c_{\text{float}} * (1 \ll (12 - q)) + 0.5),$$

2. Computational speedup of Inverse Modified Discrete Cosine Transform (IMDCT)

In order to speed-up IMDCT calculation, the simplified multiplication by transform coefficients is used.

Case IDMCT on 36 and 12 points. The transform coefficients, with absolute values smaller than 1, are represented in 32.15 format. For multiplication by this coefficients, formula (1.4) is used.

For coefficients with absolute values greater than 1, formula (1.6) is used.

Case IDMCT on 64 points (synthesis function). All transform coefficients have absolute value smaller than 1, and represented in 32.15 format. For this case, formula (1.4) is used.

Note: In high precision mode, the more precise formula (1.2) is used for all IDMCT functions.

3. Computational speedup for final windowing operation.

To generate one output sound sample in 16 bit PCM format, it is necessary to calculate convolution of samples from delay line with window coefficients.

For float data representation, the convolution loop appears as

```
for(sum=0, j=0; j<16; j++)
    sum += WindowTable[i+32*j]*line[(pos+j*64+i+(j&1)*32)&1023];
```

(3.1)

where WindowTable[512] – array of window coefficients,

pos - current position in the delay line, i - number of output samples in block of 32 samples.

The speed up is achieved by calculation of output samples in following ways:

(i) Scaled transposed window table is used:

```
WindowTableST[n] = Fix(WindowTable[i+32*j])>>q;
```

where `Fix()` corresponds (1.1) with `nBitsFraction = 24`, `n = i+32*j`, for each `i=0..31` index `j=0..15`, which provides consecutive access to array elements. Since factors of a window with indexes `j=7,8` can have absolute value greater than 1, the value `q` is obey to the rule: if `j=7` or `j=8`, `q = 9`, else `q = 8`

(ii) Optimization of a convolution loop

The convolution loop is a sequence of operators of the form

```
sum +=line[(r+g)&1023])*(*Pn_WindowTableST++)>>m;
```

where

`Pn_WindowTableST` is a pointer to the scaled transposed window table,

`r = pos + i`, and

`g = j*64+(j&1)*32`,

To provide true multiplication result, we use `m = 6` for `j=7,8`, else `m = 7`

(iii) Reduced window table for low precision mode

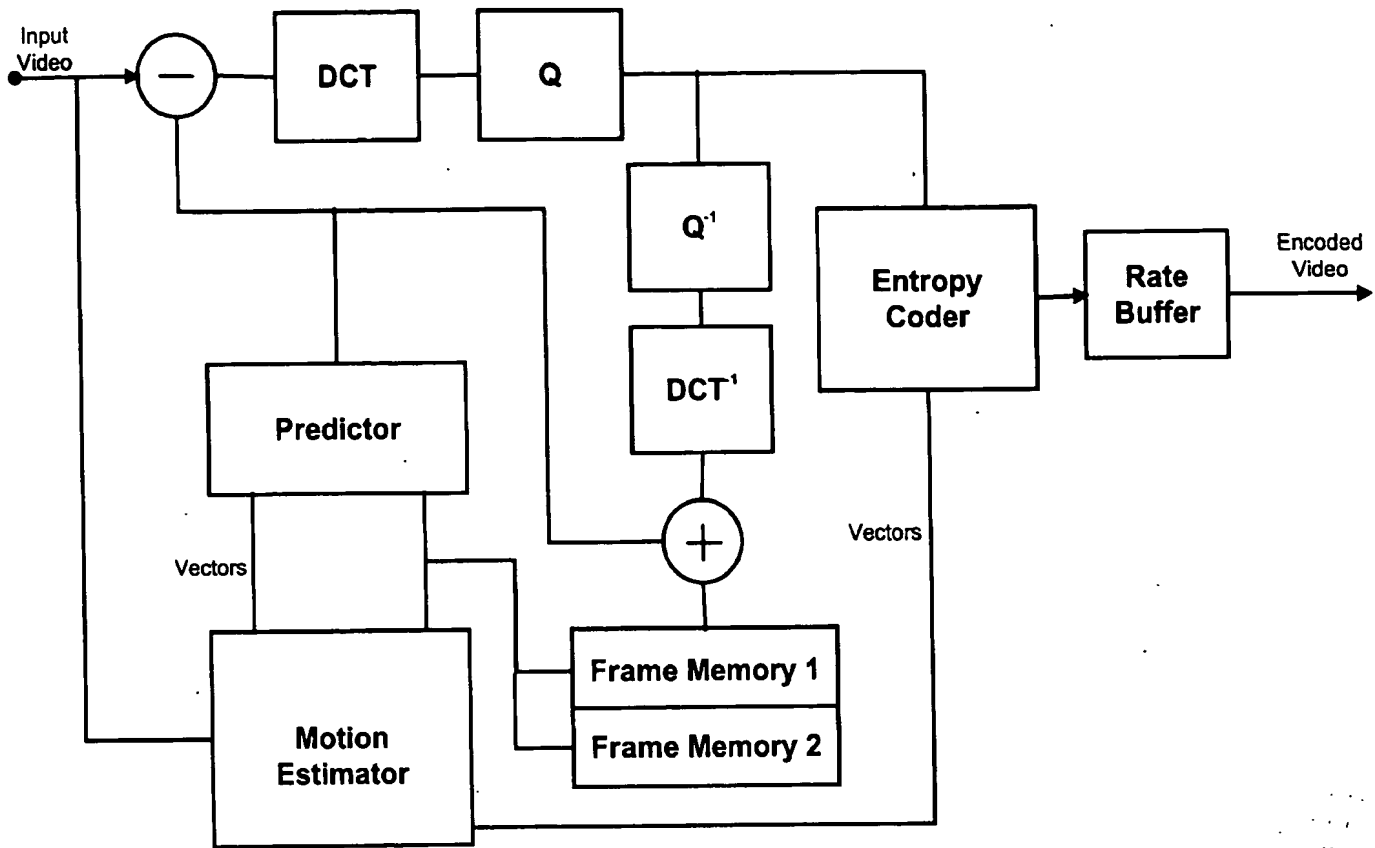
In (3.1), sum of items with number `j=0,1,2` and `j=12,13,14,15` are eliminated from calculation due to their small impact to the result (because of small window coefficients).

(iv) For high precision, 16 groups of window table items for each index `i` are normalized and have an exponent value, which is constant value inside group.

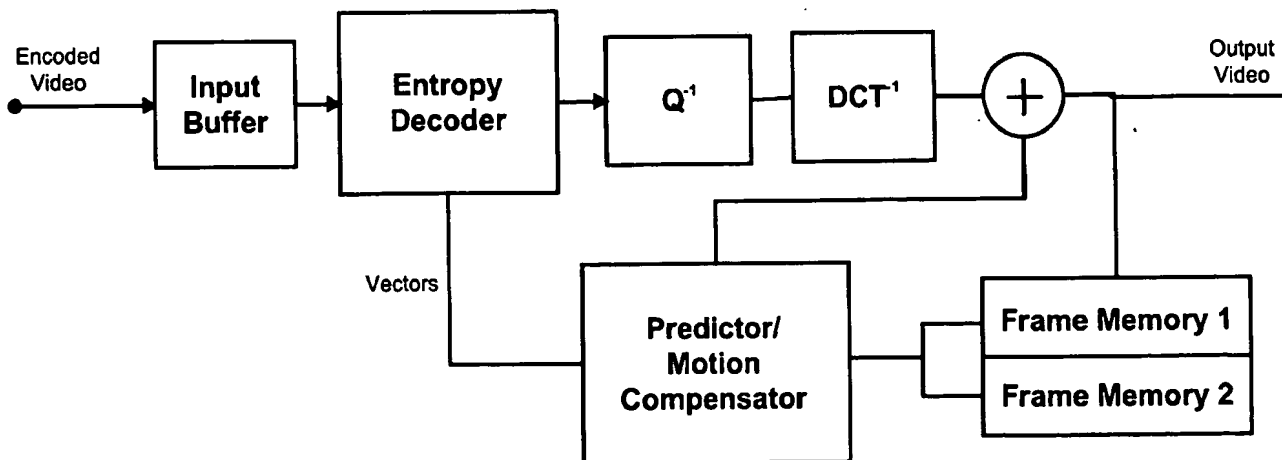
Then, the convolution loop is organized in sequence of the operators of the form

```
S[j] = line[(r+g)&1023])*(*Pn_WindowTableST++)>>7;
```

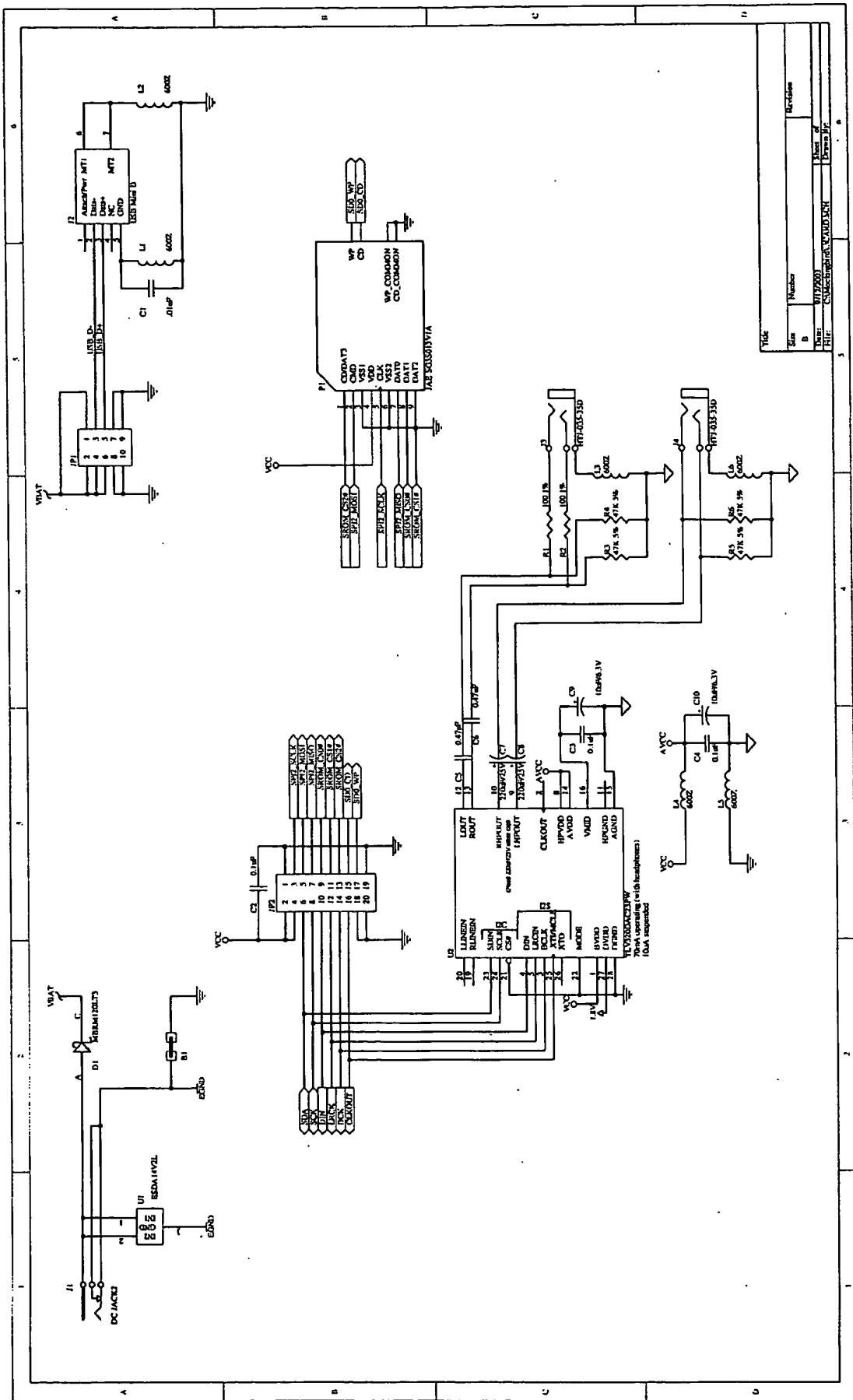
The final summation is made with shifts, which depend on values of exponents.



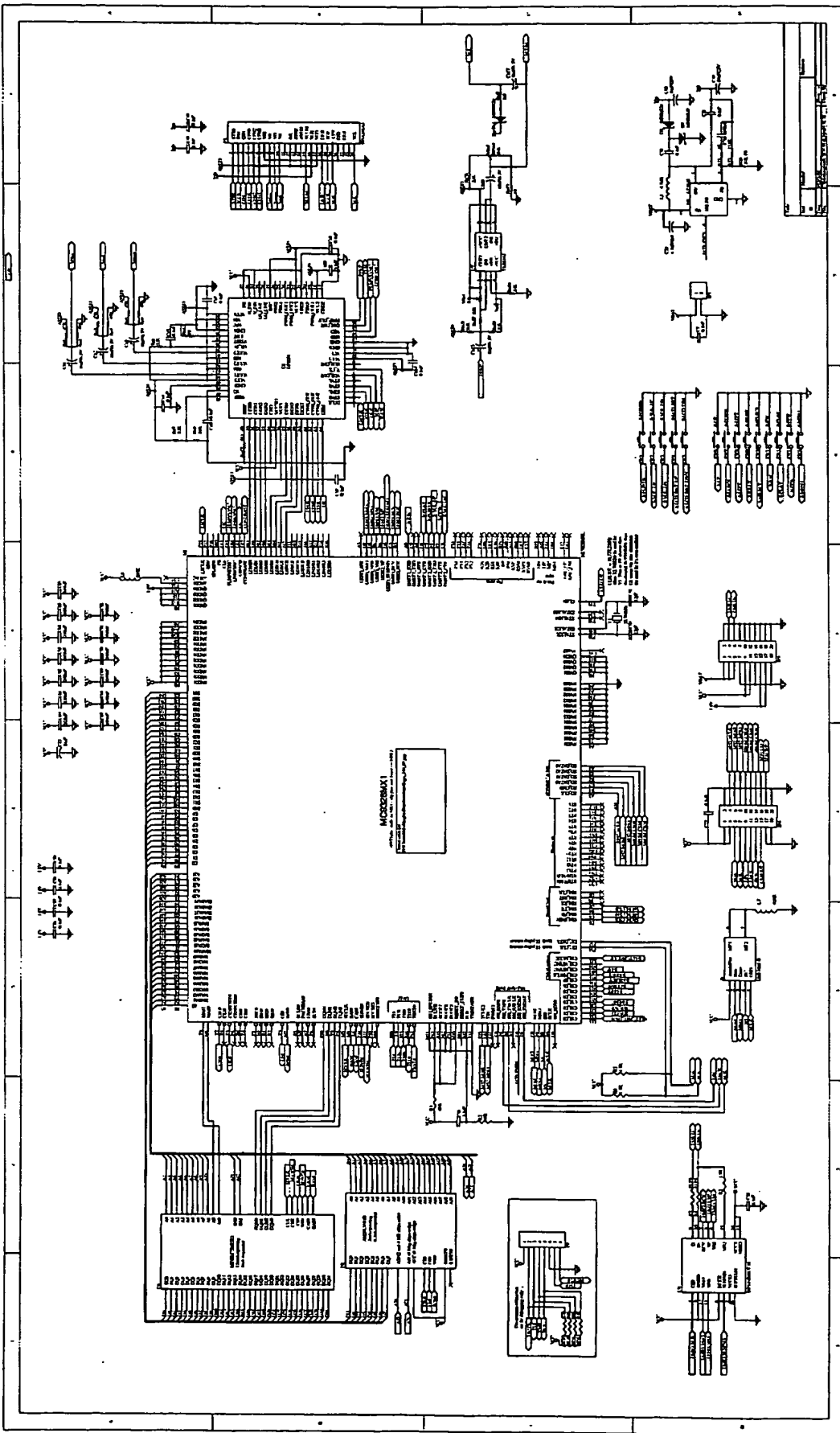
HHE Video Encoder Block Diagram



HHE Video Decoder Block Diagram



Title		Revision	
Rev	Number	Rev	Number
B	001	B	001
Date: 01/17/2001		Drawn By: [Signature]	
File: C:\Projects\usb2serial\usb2serial.sch		Sheet of: 1	



HHE Audio/Video Synchronization

HHE Audio/Video (AV) synchronization is implemented as follows:

- Each decompressed video frame is assigned a unique id (0,1,2,3,...).
- Each audio packet (containing 1152 audio samples) is also assigned a unique id (0,1,2,3,...).
- The AV sync code monitors the ids of the latest rendered video frame and audio packet.
- Every time a video interrupt occurs, these ids are recalculated into real time stamps.
- The AV sync code compares these time stamps and determine whether next video frame must be repeated (shown twice) or dropped (skipped).
- The audio stream is never adjusted. That means only video frames can be skipped or repeated to fit current audio position.

Specifically the procedure which takes place at each video interrupt is:

```
video_time_stamp = just_rendered_video_frame_id / video_frames_per_second1  
audio_time_stamp = latest_audio_id / audio_packets_per_second2
```

```
difference = audio_time_stamp - video_time_stamp
```

```
if (difference > +one_frame_duration_time)  
    skip next video frame  
else if (difference < -one_frame_duration_time)  
    repeat current video frame
```

The code can be found in file "Player.c". See video_interrupt function.

¹ Value of video_frames_per_second comes from AVI header

² Value of audio_packets_per_second is normally $44100/1152 = 38.28125$ (samples_per_sec/samples_per_packet)

HHE AVI Files

The AVI file is just a container for any number of data streams of any kind. The main parts of AVI file are:

1. The main AVI header. It always contains a stamp ("RIFF") and overall file size (for streaming). It also describes general info on the file, such as a number of streams stored in it, streams data sizes, whether the file contains an index, offset at which data streams begin, etc...
2. An optional index can be present in the avi file. It contains an entry for each data chunk (see below) describing its type and position in the file. The index is located at the very end of the file, after the data streams.
3. Each data stream format is described by its own stream header. Video stream header is actually BITMAPINFOHEADER structure (width, height, bits per pixel, compression type (HHE0 or HHE1)). Audio stream header is actually WAVEFORMATEX structure (audio format (MP3), number of channels, samples per second).
4. After all the headers, data streams begin. Data is organized in chunks. Each chunk belongs to a stream and contains a header and actual data. The header contains the stream number this chunk belongs to (usually 01 - video, 00 - audio), stream type code ("dc" - compressed video, "wb" - compressed audio), and chunk's size in bytes.

Therefore the overall layout of data is as follows:

```
01wb<chunk1 size> <- header
....chunk 1 data... <- data
00dc<chunk2 size>
....chunk2 data...
01wb<chunk3 size>
....chunk3 data...
00dc<chunk4 size>
....chunk4 data
etc...
```

There can be other types of data chunks rather than video and audio. For example, if video color format is 8 bits per pixel or less, than special palette chunk can present. But this is not our case. Note that 2 video chunks never goes one by one - there's always 1 audio chunk between them (even of zero size). As I understand, this is not a part of standard but a property of VirtualDUB. Same goes to audio chunks.

Each video chunk contains 1 compressed video frame exactly (see below on this, regarding b-frames). Each audio chunk contains either 2 or 3 audio packets (each packet is 1152 samples, when decompressed).

B-frames

When compressing with b-frames, we break the rule that each video frame is stored in its own chunk. It stores several video frames in one chunk. We also insert large amount of empty (zero length) video chunks in the stream to isolate audio chunks. So the overall layout of data streams is as follows:

<audio chunk>
<big video chunk, containing 4 frames I-P-B-B>
<audio chunk>
<empty video chunk>
<audio chunk>
<empty video chunk>
<audio chunk>
<empty video chunk>
...

This wastes actually a lot of space, since even an empty chunk contains a header and is contained in the index. This is a limitation of Video for Windows drivers. We are going to eliminate this by applying some sort of post-processing utility to avi file that will isolate each video frame in its own chunk and drop all the empty chunks.

v20030918 UI spec

Requested: SBP team

Author: C. Page, A. Dolgoborodov

Implementation: SPB team

Status: *not implemented*

Document status: HHE video playback spec is reasonably complete. MP3 and Slideshow UI incomplete.

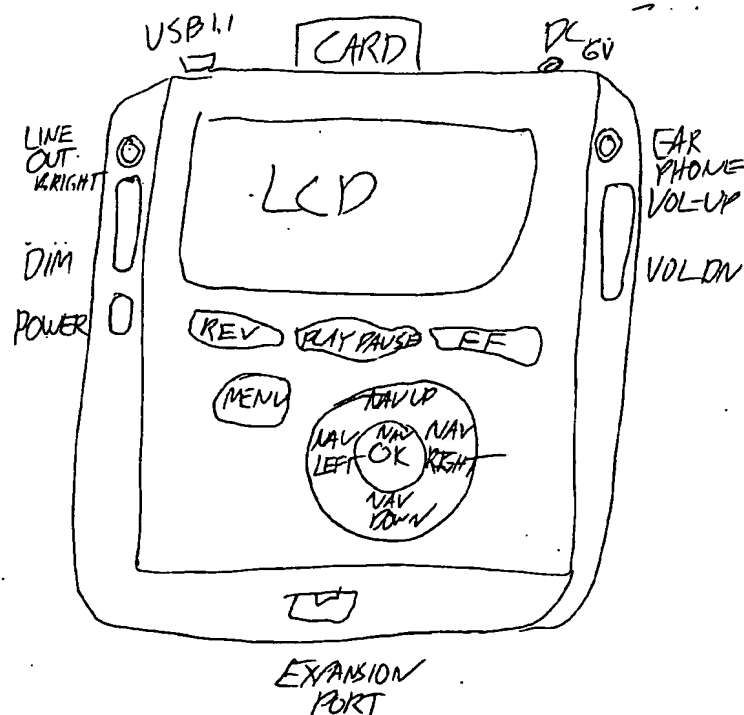
ABOUT THIS DOCUMENT

User states or modes are "quoted".

Button presses are in [square brackets]

CONTROLS

The player has 15 buttons:



DIM, BRIGHT, POWER, VOL-UP, VOL-DOWN, MENU, PLAY, FF, REV, NAV-LEFT, NAV-RIGHT, NAV-DOWN, NAV-UP, NAV-OK,

There are a number of player states. The player processes button push/release events, and some other hardware events. The player response to an event depends on its state.

VISUAL FEEDBACK

UI designers will be able to count on graphic thermometer sliders superimposed on moving video to give feedback for volume and brightness.

UI designers will be able to include compressed bitmaps for UI elements, icons, and menu screens.

The format for icons will include a transparent color.

Optional: A simple animation language may also be provided- this could be HHE format AVI, or Animated GIF (subject to IP check) or FLASH animation. (Not likely for 2003).

AUDIBLE FEEDBACK

There should be a characteristic ZVue startup sound.

Audible button feedback has two styles. Click for commands executed.

A "thud" sound for buttons pressed out of context.

Whenever a thud sound appears in UI testing, the UI design should be questioned or changed to find a way to meet the user's expectations.

A jpg viewer will also be needed for displaying digital photos. (This has not been identified as a duty of the St Petersburg team, but it will happen.)

PORTS

USB

The player may respond to a connected USB port by displaying a USB connection icon and being unresponsive to buttons aside from power, which can be used to turn it on or off.

SD Card

Upon insertion, goes to state "Media Insertion" and starts playing.

STATES

Off

The initial state for the player is "OFF", that is everything is down. The only way to get from this state is by pressing the "POWER" button [Powering ON] or by inserting a media card [Media Insertion].

ZVUE Welcome Screen

After a momentary 2 second display of the ZVUE welcome graphic and Distinctive Zvue startup sound, we return to the next expected operation.

Powering ON

On "POWER pushed" event, the ZVUE Welcome Screen is temporarily displayed.

If media is present followed by the Media menu.

Else followed by the Player Menu.

Media Insertion

The ZVUE Welcome Screen is temporarily displayed.

On "Card inserted" event, the player checks the card type.

Goes to Firmware Update if it is an update card.

Goes to application from the card if there is an application.

Goes to Media Menu Temporary if it is a media card.

Media Menu Temporary

The Media Menu is displayed offering a chance to navigate to other options.

After Timeout of only 6 seconds the media will start playing unless other media menu controls were used.

If buttons are pressed, the Timeout changes to "After 3 minutes, go OFF".

Player menu

The user is asked to insert a card, or to choose an item from the menu. The menu is

- Screen savers (*disabled*)

- Settings (includes text color and style and settings associated with mp3 and jpeg playback)

- Resume (If the player was powered OFF or paused part way through the same media that is still inserted, a resume option should appear.)

Timeout: 60 seconds transition to OFF.

Media menu

Check the media type. In the case that a writable

Sd or mmc card is found to contain both h2e media and mp3 or jpeg data
go to state "Media Choice Menu".

Timeout: 60 seconds transition to OFF.

Media menu is a short animation (may be empty), followed by a menu background picture with menu items displayed. The first menu item is active. All menu items point to video chapters.

After a period of inactivity, the menu animation restarts.
The [menu] button from media menu starts Player Menu (see above).

If the media contains more than one track, the first one will be selected and this will be visually apparent. Pressing [Play] will start that media playing. The [<<] and [>>] buttons will change the selected feature. Navigation buttons will allow moving around the UI.

PlayingHHE

When HHE AVI media cards are present, and the play function has started. This is the state the user will spend the most time in and be most attentive to.

POWER

Goes to "Off". If the media is longer than 5 minutes, the position it was playing at is stored.

MENU goes to the "MediaMenu"

PLAY goes to "PlayingHHEPause",

FF, Fast Forward feature of "PlayingHHE" state.

REV, Skip back feature of "PlayingHHE" state

NAV-LEFT, Previous Video "Chapter"

NAV-RIGHT, Next Video "Chapter"

NAV-UP, Slow Motion feature enabled or disabled.

NAV-OK, Sound continues, but Playing menu on screen. Goes to state "PlayingHHE-MENU"

The NAV DOWN button enables the AB REPEAT feature.

Here is the AB/Repeat state table. (These states are sub-states of PLAYING)

PLAYING

Shows the video normally. Moves to the next track when done.
Pressing A/B repeat moves it to state Playing-A at that position.

PLAYING-A

When the video auto-repeats, it will restart at point A instead of the start.
Pressing A/B repeat moves it to state Playing-AB at that position.

PLAYING-AB

When the video auto-repeats, it will restart at point A instead of the start and go to point B instead of the end.
Pressing A/B repeat moves it to state Playing-Autorepeat.

Media Choice Menu

It is possible that people will combine content HHE downloads with their own MP3 and JPEG content. Only in that case is this state necessary.

Displays options as available on the card.

Video (takes you to the media menu for that content.)

Slide Show (starts playing Audio files and Pictures together- see state "Slide Show")

Music (navigates folders of MP3 files- see state "MP3 Player")

Pictures (navigates pictures- see state "JPEG Viewer")

Slide Show Menu

Software Prepares two playlists. The Audio Playlist, and the Photo Playlist.

If a playlist file is on the card it may use that to determine the order of audio and video files. Otherwise both playlists are in breadth-first recursive order through the folders with the files sorted in the most natural order possible (To be specified).

[play] takes you to state *Slide Show Playing*.

Slide Show Playing

The [<<.] [play] [>>] buttons will effect the music playback.

The direction keys will effect the photo selection. [Right] and [Left] will go to previous and next picture.

[menu] will bring up the "slideshow menu".

[select] will bring up the "slide menu"

Slide Menu

Displays the current slide. If possible it should display the whole slide- then zoom in slightly. If the slide is vertical format

The [<<.] [play] [>>] buttons will effect the music playback.

The 4 direction keys will effect the photo position, panning the photo in the chosen direction until the edge is reached where it stops, making a "thud" sound.

[menu] will zoom out more. If totally zoomed out will offer "Slide Show Playing" options.

[select] will zoom in more. If totally zoomed in will offer "Slide Menu Detail"

Timeout: go to next slide in the sequence after adjustable time determined in settings.

Slide Menu Detail

Offers these choices by text or icon.

- SlideShow Delay (amount of time before slide advance.)
- Rotate picture
- Gamma Adjust
- Special Effects
- Crop here
- Choose animation
- Choose soundtrack

QUESTIONS

- Chose between detailed controls for mp3 or jpeg viewer?
- Have a play/pause status menu shift in slideshow player?
- Define playlist?
- Save staus or setup on sd card?

JPEG Viewer

When there are no MP3's the player will behave as above except with no music.

MP3 Player

Menu structure will show one directory of the FAT file system.
Only folders with displayable content will be shown.

THE BASICS

Menu navigation

The NAV-* keys control the selection of a menu item. On "OK" transition is made to metu item selected.

TBD: details on navigation keys

Volume and brightness control

Volume control range:	-73..+6 dB
Volume control granularity:	1 dB
Volume level display timeout:	5 sec
Volume level display:	horizontal bar at the bottom of the screen

On reset, the audio level and screen brightness are set to default values. The default values are stored in the flash.

Pressing the audio level control button in any player state except "OFF" result in current level being displayed in the bottom of the screen. Subsequent pressures on volume buttons change audio level by 1 dB. After volume control buttons are untouched for 5 sec, the volume level bar disappear.

Brightness control

DIM and BRIGHT move the player up and down through at least 5 brightness settings.

No visual indicator on screen except for actual screen brightness change. At the dimmest setting, the display is Off—useful for conserving batteries when only audio is desired. (In this case, software should do less video work). At Display Off, any brightness input will display.

(Note: If display is off while audio is playing, the volume indicator should appear on the screen when the Volume rocker button is pressed for the sake of consistency, and user convenience.)

Playback control

When inside the video, STOP button starts media menu.

When playing:

- PLAY button pauses the player
- REV button restarts the chapter
- FF button: audio off, video playback is 2X

When paused:

- PLAY resumes from pause
- REV goes to the beginning of the chapter, doesn't resume from the pause
- FF audio off, video playback is 2X (approx.)

When FF 2X

- PLAY audio on, normal speed
- REV same as PLAY
- FF Audio off, video at 6X speed. Player does it by skipping B and, if necessary, P frames. This can result in the loss of continuity.

Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/US04/032296

International filing date: 29 September 2004 (29.09.2004)

Document type: Certified copy of priority document

Document details: Country/Office: US
Number: 60/507,185
Filing date: 29 September 2003 (29.09.2003)

Date of receipt at the International Bureau: 17 November 2004 (17.11.2004)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

☐ **BLACK BORDERS**

☒ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**

☒ **FADED TEXT OR DRAWING**

☒ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**

☐ **SKEWED/SLANTED IMAGES**

☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**

☒ **GRAY SCALE DOCUMENTS**

☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**

☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**

☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.